A Nonmonotone Analysis with the Primal-Dual Approach: online routing of virtual circuits with unknown durations

Guy Even* Moti Medina*

Abstract

We address the question of whether the primal-dual approach for the design and analysis of online algorithms can be applied to nonmonotone problems. We provide a positive answer by presenting a primal-dual analysis to the online algorithm of Awerbuch et al. [AAPW01] for routing virtual circuits with unknown durations.

1 Introduction

The analysis of most online algorithms is based on a potential function (see, for example, [AAP93, AKP+97, AAF+97, AAPW01] in the context of online routing). Buchbinder and Naor [BN09] presented a primal-dual approach for analyzing online algorithms. This approach replaces the need to find the appropriate potential function by the task of finding an appropriate linear programming formulation.

The primal-dual approach presented by Buchbinder and Naor has a monotone nature. Monotonicity means that: (1) Variables and constraints arrive in an online fashion. Once a variable or constraint appears, it is never deleted. (2) Values of variables, if updated, are only increased. We address the question of whether the primal-dual approach can be extended to analyze nonmonotone algorithms[1].

An elegant example of nonmonotone behavior occurs in the problem of online routing of virtual circuits with unknown durations. In the problem of routing virtual circuits, we are given a graph with edge capacities. Each request r_i consists of a source-destination pair (s_i, t_i). A request r_i is served by allocating to it a path from s_i to r_i. The goal is to serve the requests while respecting the edge capacities as much as possible. In the online setting, requests arrive one-by-one. Upon arrival of a request r_i, the online algorithm must serve r_i. In the special case of unknown durations, at each time step, the adversary may introduce a new request or it may terminate an exiting request. When a request terminates, it frees the path that was allocated to it, thus reducing the congestion along the edges in the path. The online algorithm has no knowledge of the future; namely, no information about future requests and no information about when existing requests will end. Nonmonotonicity is expressed in this online problem in two ways: (1) Requests terminate thus deleting the demand to serve them. (2) The congestion of edges varies in a nonmonotone fashion; an addition of a path increases congestion, and a deletion of a path decreases congestion.

Awerbuch et al. [AAPW01] presented an online algorithm for online routing of virtual circuits when the requests have unknown durations. In fact, their algorithm resorts to rerouting to obtain a logarithmic competitive ratio for the load. Rerouting means that the path allocated to a request is not fixed and the algorithm may change this path from time to time. Hence, allowing rerouting increases the nonmonotone characteristics of the problem.

*School of Electrical Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel. {guy,medinamo}@eng.tau.ac.il.
M.M was partially funded by the Israeli Ministry of Science and Technology.

[1]The only instance we are aware of in which the primal-dual approach is applied to nonmonotone variables appears in [BFGN11]. In this instance, the change in the dual profit, in each round, is at least a constant times the change in the primal profit. In general, this property does not hold in a nonmonotone setting.

We present an analysis of the online algorithm of Awerbuch et al. [AAPW01] for online routing of virtual circuits with unknown durations. Our analysis uses the primal-dual approach, and hence we show that the primal-dual approach can be applied in nonmonotone settings.

2 Problem Definition

2.1 Online Routing of Virtual Circuits with Unknown Durations

Let $G = (V, E)$ denote a directed or undirected graph. Each edge e in E has a capacity $c_e \geq 1$. A routing request r_k is a 4-tuple $r_k = (s_k, d_k, a_k, b_k)$, where (i) $s_k, d_k \in V$ are the source and the destination of the kth routing request, respectively, (ii) $a_k \in \mathbb{N}$ is both the arrival time and the start time of the request, and (iii) $b_k \in \mathbb{N}$ is the departure time or end time of the request. Let Γ_k denote the set of paths in G from s_k to d_k. A request r_k is served if it is allocated a path in Γ_k.

Let $[N]$ denote the set $\{0, \ldots, N\}$. The input consists of a sequence of events $\sigma = \{\sigma_t\}_{t\in[N]}$. We assume that time is discrete, and event σ_t occurs at time t. There are two types of events: (i) An *arrival* of a request. When a request r_k arrives, we are given the source s_k and the destination d_k. Note that the arrival time a_k simply equals the current time t. (ii) A *departure* of a request. When a request r_k departs there is no need to serve it anymore (namely, the departure time b_k simply equals the current time t).

The set of active requests at time t is denoted by *Alive*$_t$ and is defined by

$$\mathit{Alive}_t \triangleq \{r_k \mid a_k \lneqq t \leq b_k\} .$$

An *allocation* is a sequence $A = \{p_k\}_k$ of paths such that p_k is a path from the source s_k to the destination d_k of request r_k. Let *paths*$_t(e, A)$ denote the number of requests that are routed along edge e by allocation A at time t, formally:

$$\mathit{paths}_t(e, A) \triangleq |\{p_k : e \in p_k \text{ and } r_k \in \mathit{Alive}_t\}| .$$

The *load* of an edge e at t is defined by

$$\mathit{load}_t(e, A) \triangleq \frac{\mathit{paths}_t(e, A)}{c_e} .$$

The *load* of an allocation A at time t is defined by

$$\mathit{load}_t(A) \triangleq \max_{e\in E} \mathit{load}_t(e, A) .$$

The *load* of an allocation A is defined by

$$\mathit{load}(A) \triangleq \max_{t} \mathit{load}_t(A) .$$

An algorithm computes an allocation of paths to the requests, and therefore we abuse notation and identify the algorithm with the allocation that is computed by it. Namely, $\text{ALG}(\sigma)$ denotes the allocation computed by algorithm ALG for an input sequence σ.

In the online setting, the events arrive one-by-one, and no information is known about an event before its arrival. Moreover, (1) the length N of the sequence of events is unknown; the input simply stops at some point, (2) the departure time b_k is *unknown* (and may even be determined later by the adversary), and (3) the online algorithm must allocate a path to the request as soon as the request arrives.

The *competitive ratio* of an online algorithm ALG with respect to $N \in \mathbb{N}$, and a sequence $\sigma = \{\sigma_t\}_{t\in[N]}$ is defined by

$$\rho(\text{ALG}(\sigma)) \triangleq \frac{\mathit{load}(\text{ALG}(\sigma))}{\mathit{load}(\text{OPT}(\sigma))} ,$$

where $\text{OPT}(\sigma)$ is an allocation with minimum load. The *competitive ratio* of an online algorithm ALG is defined by

$$\rho(\text{ALG}) \triangleq \sup_{N \in \mathbb{N}} \max_{\sigma} \rho(\text{ALG}(\sigma)) \, .$$

Note that since every request has a unit demand, we may assume that $c_e \geq 1$ for every edge $e \in E$.

2.2 Rerouting

In the classical setting, a request r_k is served by a fixed single path p_k throughout the duration of the request. The term *rerouting* means that we allow the allocation to change the path p_k that serves r_k. Thus, there are two extreme cases: (i) no rerouting at all is permitted (classical setting), and (ii) total flexibility in which, a new allocation can be computed in each time step.

Following the paper by Awerbuch et al. [AAPW01], we allow the online algorithm to reroute each request at most $O(\log |V|)$ times. In the analysis of the competitive ratio, we compare the load of the online algorithm with the load of an optimal (splittable) allocation with total rerouting flexibility. Namely, the optimal solution recomputes a minimum load allocation at each time step, and, in addition may serve a request by a convex combination of paths.

3 The Online Algorithm ALG

In this section we present the online algorithm ALG that is listed in Algorithm 1. Thus algorithm is equivalent to the algorithm presented in [AAPW01].

The algorithm maintains the following variables.

1. For every edge e a variable x_e. The value of x_e is exponential in the load of edge e.

2. For every request r_k a variable z_k. The value of z_k is the complement of the "weight" of the path p_k allocated to r_k at the time the path was allocated.

3. For every routing request r_k, and for every path $p \in \Gamma_k$ a variable $f_k(p)$. The value of $f_k(p)$ indicates whether p is allocated to r_k. That is, the value of $f_k(p)$ equals 1 if path p is allocated for request r_k, and 0 otherwise.

The algorithm ALG consists of the following 5 procedures: (1) MAIN, (2) ROUTE, (3) DEPART, (4) UNROUTE, and (5) MAKEFEASIBLE.

The MAIN procedure begins with initialization. For every $e \in E$, x_e is initialized to $\frac{1}{4m}$, where $m = |E|$. For every $k \in [N]$, z_k is initialized to zero. For every $k \in [N]$, and for every path p, $f_k(p)$ is initialized to zero. Since the number of z_k and $f_k(p)$ variables is unbounded, their initialization is done in a "lazy" fashion; that is, upon arrival of the kth request the corresponding variables are set to zero.

The main procedure MAIN proceeds as follows. For every time step $t \in [N]$, if the event σ_t is an arrival of a request, then the ROUTE procedure is invoked. Otherwise, if the event σ_t is a departure of a request, then the DEPART procedure is invoked.

The ROUTE procedure serves request r_k by allocating a "lightest" path p_k in the set Γ_k (recall that Γ_k denotes the set of paths from the source s_k to the destination d_k). The allocation is done by two actions. First, the allocation of p_k to request r_k is indicated by setting $f_k(p_k) \leftarrow 1$. Second, the loads of the edges along p_k are updated by increasing the variables x_e for $e \in p_k$. The variable z_k equals the "complement" weight of the allocated path p_k. Note that this complement is with respect to half the weight of the path before its update.

The DEPART procedure "frees" the path that is allocated for p_k, by calling the UNROUTE procedure. The UNROUTE procedure frees p_k by nullifying $f_k(p_k)$ and z_k, and by decreasing the edge variables x_e for the edges along p_k. The freeing of p_k, decreases the load along the edges in p_k. As a result of this decrease, it may happen that a path allocated to an alive request might be very heavy compared to a lightest path. In such a case, the request should be rerouted. This is why the MAKEFEASIBLE procedure is invoked after the UNROUTE procedure.

Rerouting is done by the MAKEFEASIBLE procedure. This rerouting is done by freeing a path and then routing the request again. Requests with improved alternative paths are rerouted.

The listing of the online algorithm ALG appears in Algorithm 1.

Algorithm 1 ALG: Online routing algorithm. The input consists of (1) a graph $G = (V, E)$ where each $e \in E$ has capacity c_e, and (2) a sequence of events $\sigma = \{\sigma_t\}_{t \in [N]}$.

```
MAIN(σ_t)
1: $\forall k \in [N] : z_k \leftarrow 0$.
2: $\forall e \in E : x_e \leftarrow \frac{1}{4m}$, where $m = |E|$.
3: $\forall r_k \in [N] \; \forall p : f_k(p) \leftarrow 0$.
4: Upon arrival of event $\sigma_t$ do
5:     if $\sigma_t$ is an arrival of request $r_k$ then Call ROUTE($r_k$).
6:     else ($\sigma_t$ is an departure of request $r_k$) Call DEPART($r_k$).

ROUTE(r_k)
1: Find the "lightest" path: $p_k \leftarrow$ argmin$\{\sum_{e \in p'} \frac{x_e}{c_e} \mid p' \in \Gamma_k\}$.
2: $z_k \leftarrow 1 - \frac{1}{2} \cdot \sum_{e \in p_k} \frac{x_e}{c_e}$.
3: Route $r_k$ along $p_k$: $f_k(p_k) \leftarrow 1$.
4: for all $e \in p_k$ do
5:     $x_e \leftarrow x_e \cdot \lambda_e$ where $\lambda_e \triangleq \left(1 + \frac{1}{4c_e}\right)$. {Update edge "load"}

DEPART(r_k)
1: Call UNROUTE($r_k$).
2: Call MAKEFEASIBLE($x, z$).

UNROUTE(r_k)
1: Free variables: $z_k, f_k(p_k)$.
2: for all $e \in p_k$ do
3:     $x_e \leftarrow x_e / \lambda_e$ where $\lambda_e \triangleq \left(1 + \frac{1}{4c_e}\right)$. {Update edge "load"}

MAKEFEASIBLE(x, z)
1: $\forall r_j \in Alive_t$ if $\exists p \in \Gamma_j \; : z_j + \sum_{e \in p} \frac{x_e}{c_e} < 1$ then
2:     Call UNROUTE($r_j$).
3:     Call ROUTE($r_j$).
```

4 Primal-Dual Analysis of ALG

In this section we prove that the load on every edge is always $O(\log |V|)$, and that each request is rerouted at most $O(\log |V|)$ times. We refer to an input sequence σ as *feasible* if there is an allocation A, such that for all requests that are alive at time t, it holds that *load*$_t(A) \le 1$. The following theorem holds under the assumption that the input sequence σ is feasible. Note that the removal of this assumption increases the competitive ratio only by a constant factor by standard doubling techniques [AAPW01].

Theorem 1 ([AAPW01]). *If the input sequence σ is feasible and assuming that $c_e \ge 1$, then* ALG *is:*

1. *An $O(\log |V|)$-competitive online algorithm.*
2. *Every request is rerouted at most $O(\log |V|)$ times.*

We point out that the allocation computed by ALG is *nonsplittable* in the sense that at every given time each request is served by a single path. The optimal allocation, on the other hand, is both totally flexible and *spilttable*. Namely, the optimal allocation may reroute all the requests in each time step, and, in addition, may serve a request by a convex combination of paths.

$$\underline{\textsc{p-lp}(t)}: \quad \min \sum_{r_k \in \mathit{Alive}_t} z_k + \sum_{e \in E} x_e \quad \text{s.t.}$$

$$\forall r_k \in \mathit{Alive}_t\ \forall p \in \Gamma_k : z_k + \sum_{e \in p} \frac{x_e}{c_e} \geq 1 \text{ (Covering Constraints.)}$$

$$x \geq \vec{0}$$

(I)

$$\underline{\textsc{d-lp}(t)}: \quad \max \sum_{r_k \in \mathit{Alive}_t} \sum_{p \in \Gamma_k} f_k(p) \quad \text{s.t.}$$

$$\forall e \in E : \frac{1}{c_e} \cdot \sum_{r_k \in \mathit{Alive}_t} \sum_{\{p | p \in \Gamma_k, e \in p\}} f_k(p) \leq 1 \text{ (Capacity Constraints.)}$$

$$\forall r_k \in \mathit{Alive}_t : \sum_{p \in \Gamma_k} f_k(p) = 1 \text{ (Demand Constraints.)}$$

$$f \geq \vec{0}$$

(II)

Figure 1: (I) The primal LP, $\textsc{p-lp}(t)$. (II) The dual LP, $\textsc{d-lp}(t)$.

The rest of the proof is as follows. We begin by formulating a packing and covering programs for our problem in Section 4.1. We then prove Lemma 1 in Section 4.2. We conclude the analysis with the proof of Theorem 1 in Section 4.3

4.1 Formulation as an Online Packing Problem

For the sake of analysis, we define for every prefix of events $\{\sigma_j\}_{j=1}^t$ a *primal* linear program $\textsc{p-lp}(t)$ and its *dual* linear program $\textsc{d-lp}(t)$. The primal LP is a *covering* LP, and the dual LP is a *packing* LP. The LP's appear in Figure 1.

The variables of the LPs correspond to the variables maintained by ALG, as follows. The covering program $\textsc{p-lp}(t)$ has a variable x_e for every edge $e \in E$, and a variable z_k for every $r_k \in \mathit{Alive}_t$. The packing program $\textsc{d-lp}(t)$ has a variable $f_k(p)$ for every request $r_k \in \mathit{Alive}_t$, and for every path $p \in \Gamma_k$. The variable $f_k(p)$ equals to the fraction of r_k's "demand" that is routed along path $p \in \Gamma_k$.

The dual LP has three types of constraints: capacity constraints, demand constrains, and sign constraints. In the fractional setting the load of an edge is defined by

$$\mathit{load}_t(e) \triangleq \frac{1}{c_e} \cdot \sum_{r_k \in \mathit{Alive}_t} \sum_{\{p | p \in \Gamma_k, e \in p\}} f_k(p) .$$

The capacity constraint in the dual LP requires that the load of each edge is at most one. The demand constraints require that each request r_k that is alive at time t is allocated a convex combination of paths.

If the dual LP is feasible, then the objective function of the dual LP simply equals the number of requests that are alive at time step t, i.e., $|\mathit{Alive}_t|$.

The primal LP has two types of constraints: covering constraints and sign constraints. The covering constraints requires that for every request r_k that is alive and for every path $p \in \Gamma_k$, the sum of z_k and the "weight" of p is at least 1. Note that the sign constraints apply only to the edge variables x_e whereas the request variables z_k are free.

Note that the assumption that σ is feasible is equivalent to requiring that the dual program $\textsc{d-lp}(t)$ is feasible for every t.

4.2 Bounding the Primal Variables

In this section we prove that the primal variables x_e are bounded by a constant, as formalized in the following Lemma.

Lemma 1. *If σ_t is an original event, then*

$$\forall e \in E \ : x_e^{(t)} \leq 3 \, .$$

The proof of Lemma 1 is based on a few lemmas that we prove first.

Notation. Let $x_e^{(t)}, z_k^{(t)}$ denote the value of the primal variables x_e, z_k before event σ_t is processed by ALG. Let P_t denote the objective function's value of P-LP(t), formally:

$$P_t \triangleq \sum_{r_k \in \mathit{Alive}_t} z_k^{(t)} + \sum_{e \in E} x_e^{(t)} \, .$$

Let $\Delta_t P \triangleq P_{t+1} - P_t$.

Note that P_t refers to the value of P-LP(t) at the beginning of time step t. The definition of *Alive*$_t$ implies that the constraints and variables of P-LP(t) are not influenced by the event σ_t (this happens only for P-LP$(t+1)$). Hence the variables in the definition of P_t are indexed by time step t.

Dummy events. The procedure ROUTE is invoked in two places: (i) in Line 5 of MAIN as a result of an arrival of a request, or (ii) in Line 3 of MAKEFEASIBLE. To simplify the discussion, we create "dummy" events each time the MAKEFEASIBLE procedure reroutes a request. Dummy events come in pairs: first a dummy departure event for request r_k is introduced, and then a dummy arrival event for a "continuation" request r_k is introduced. The combination of original events and dummy events describes the execution of ALG. The augmentation of the original input sequence of events by dummy events does not modify the optimal value of the dual LP at time steps t that correspond to original events. Hence, we analyze the competitive ratio $\rho(\text{ALG}(\sigma))$ by analyzing the competitive ratio with respect to the augmented sequence at time steps t that correspond to original events.

The following lemma follows immediately from the description of the algorithm ALG and the definition of dummy events.

Lemma 2 (Primal Feasibility). *If σ_t is an original event, then the variables $\{x_e^{(t)}\}_{e \in E} \cup \{z_\ell^{(t)}\}_{\ell \in \mathit{Alive}_t}$ constitute a feasible solution for P-LP(t).*

Proof. When an original event $\sigma_{t'}$ occurs, the MAKEFEASIBLE procedure generates dummy events at the end of the time step to guarantee that the primal variables are a feasible solution of the primal LP. Hence, if σ_t is an original event, then the primal variables at the beginning of time step t are a feasible solution for P-LP(t). □

Lemma 3. *If σ_t is an arrival of request, then $\Delta_t P < 1$.*

Proof. Assume that σ_t is an event in which request r_k arrives. In Step 2 of the ROUTE algorithm z_k is set to $1 - \frac{1}{2} \cdot \sum_{e \in p_k} \frac{x_e^{(t)}}{c_e}$. In Step 5 of the ROUTE algorithm, for every $e \in p_k$, x_e is increased by $\frac{x_e^{(t)}}{4c_e}$. All the other edge variables x_e remain unchanged. Hence,

$$\begin{aligned}
\Delta_t P =& 1 - \frac{1}{2} \cdot \sum_{e \in p_k} \frac{x_e^{(t)}}{c_e} + \sum_{e \in p_k} \frac{x_e^{(t)}}{4c_e} \\
=& 1 - \frac{1}{4} \cdot \sum_{e \in p_k} \frac{x_e^{(t)}}{c_e} \qquad (1) \\
<& 1 \, ,
\end{aligned}$$

as required. ☐

We refer to the number of requests that are routed along edge e by allocation ALG at time t by *paths*$_t(e)$.

Lemma 4. *For every t and $e \in E$,*

$$x_e^{(t)} = \frac{1}{4m} \cdot \lambda_e^{\mathit{paths}_t(e)} .$$

Proof. The proof is by induction on t. At time $t = 0$, we have $x_e^{(0)} = \frac{1}{4m}$ and *paths*$_t(e) = 0$. The proof of the induction basis for $t + 1$ depends on whether at time step t an arrival or a departure occurs. If the event does not affect edge e, then the induction step clearly holds. Assume that the event affects edge e. If a request r_k arrives at time t, then *paths*$_{t+1}(e) =$ *paths*$_t(e) + 1$ and $x_e^{(t+1)} = x_e^{(t)} \cdot \lambda_e$. If a request r_k departs at time t, then *paths*$_{t+1}(e) =$ *paths*$_t(e) - 1$ and $x_e^{(t+1)} = x_e^{(t)} / \lambda_e$. ☐

Let *Dead*$_t \triangleq \{r_k \mid b_k < t\}$. In general, it is not true that $\Delta_{a_j} P + \Delta_{b_j} P \leq 0$, however on average it is true, as stated in the following lemma.

Lemma 5. *For every t,*

$$\sum_{r_j \in \mathit{Dead}_t} \left(\Delta_{a_j} P + \Delta_{b_j} P\right) \leq 0 . \tag{2}$$

Proof. First we prove the following proposition.

Proposition 1. *Consider a set of $I = \{I_j = [\alpha_j, \beta_j]\}_{j=1}^q$ such that no two intervals share a common endpoint. Let cut(t) denote the number of intervals that contain t. Then, there is a permutation $\pi : [1, q] \to [1, q]$ such that*

$$\forall j \in [1, q] \ : \ \mathit{cut}(\alpha_j) = \mathit{cut}(\beta_{\pi(j)}) . \tag{3}$$

Proof. The proof is by induction on the number of intervals. The induction basis, for $q = 1$ holds trivially because *cut*$(\alpha_1) =$ *cut*$(\beta_1) = 1$. The proof of the induction step is based on the existence of a pair $\alpha_i < \beta_j$ such that the open interval (α_i, β_j) does not contain any endpoint of the intervals in I. For such a pair, we immediately have *cut*$(\alpha_i) =$ *cut*(β_j) so we define $\pi(i) = j$ and apply the induction hypothesis.

We first show that such a pair $\alpha_i < \beta_j$ exists. We say that an interval I_m is *minimal* if $I_m \cap I_k \neq \emptyset$ implies that $I_m \subseteq I_k$. If there exists a minimal interval I_m, then set $\alpha_i = \alpha_m$ and $\beta_j = \beta_m$. In such a case since $\pi(m) = m$, we can erase I_m and proceed by applying the induction hypothesis to the remaining intervals. Note that equality of cut sizes is preserved when the interval I_m is deleted.

Consider the set of pairs of intersecting intervals without containment defined as follows

$$A \triangleq \{(i, j) \mid \alpha_j < \alpha_i < \beta_j < \beta_i\} .$$

If there is no minimal interval, the set A is not empty. Any pair $(i, j) \in A$ that minimizes the difference $(\beta_j - \alpha_i)$ has the property that the interval (α_i, β_j) lacks endpoints of intervals in I.

We can define $\pi(i) = j$. We proceed by applying the induction hypothesis on $(I \setminus \{I_j, I_i\}) \cup I_k$, where $I_k = I_i \cup I_j$. Note that equality of cut sizes is preserved when I_i and I_j are merged into one interval. ☐

The difference $\Delta_{a_j} P$ consists of two parts:

$$\Delta_{a_j} P = z_j^{(a_j+1)} + \sum_{e \in p_j} \frac{x_e^{(a_j)}}{4c_e}.$$

The difference $\Delta_{b_j} P$ consists of two parts as well:

$$\Delta_{b_j} P = -z_j^{(b_j)} - \sum_{e \in p_j} \frac{x_e^{(b_j+1)}}{4c_e}.$$

It follows that

$$\begin{aligned}\sum_{r_j \in \mathit{Dead}_t} \left(\Delta_{a_j} P + \Delta_{b_j} P\right) &= \sum_{r_j \in \mathit{Dead}_t} \sum_e \frac{1}{4c_e} \cdot \left(x_e^{(a_j)} - x_e^{(b_j+1)}\right) \\ &= \sum_{r_j \in \mathit{Dead}_t} \sum_e \frac{1}{4c_e} \cdot \left(x_e^{(a_j)} - x_e^{(b_{\pi(j)}+1)}\right),\end{aligned}$$

where π is any permutation over the set of requests. In fact, we shall use for each edge e, a different permutation $\pi = \pi(e)$ that is a permutation over the requests r_k such that $e \in p_k$.

Assume first that $\mathit{Alive}_t = \emptyset$. We later lift this assumption.

Fix an edge e. For each request r_j such that $e \in p_j$, map the duration $(a_j, b_j]$ of request r_j to the interval $[a_j + 1, b_j]$. The resulting set of intervals satisfies $\mathit{cut}(t) = \mathit{paths}_t(e)$ for every time step t. Let π denote the permutation guaranteed by Prop. 1. Then, it suffices to prove that

$$x_e^{(a_j)} - x_e^{(b_{\pi(j)}+1)} = 0. \tag{4}$$

Indeed, by Lemma 4, $4m \cdot \left(x_e^{(a_j)} - x_e^{(b_{\pi(j)}+1)}\right) = \lambda_e^{\mathit{paths}_{a_j}} - \lambda_e^{\mathit{paths}_{b_{\pi(j)}+1}}$. In addition, the property of permutation π states that $\mathit{cut}(a_j + 1) = \mathit{cut}(b_{\pi(j)})$. It follows that $\mathit{paths}_{a_j+1} = \mathit{paths}_{b_{\pi(j)}}$. But, $\mathit{paths}_{a_j} = \mathit{paths}_{a_j+1} - 1$ and $\mathit{paths}_{b_{\pi(j)}+1} = \mathit{paths}_{b_{\pi(j)}} - 1$, and Equation 4 follows.

To complete the proof, consider the requests in Alive_t. Because $a_j, b_{\pi(j)} \le t$, requests in Alive_t do not increase the difference $x_e^{(a_j)} - x_e^{(b_{\pi(j)}+1)}$. Thus $x_e^{(a_j)} - x_e^{(b_{\pi(j)}+1)} \le 0$, and the lemma follows. □

We are now ready to prove Lemma 1. Recall that Lemma 1 states that the primal variables x_e are bounded by a constant. The proof of Lemma 1 is by contradiction. In fact, we reach a contradiction to *weak duality*, that is, we show that the value of the primal solution is strictly smaller than the value of a feasible dual solution.

Proof of Lemma 1. The proof is by contradiction. Assume $x_e^{(t)} > 3$ and σ_t is an original event. Define

$$t_2 \triangleq \min\{t \mid x_e^{(t)} > 3 \text{ and } \sigma_t \text{ is an original event}\}.$$

Let t_1 be the time step for which $x_e^{(t_1)} < 1$ and $x_e^{t'} \ge 1$ for every $t' \in [t_1 + 1, t_2]$.

Define:

$$\mathit{Alive}_{\in e}(t_1, t_2) \triangleq \{r_j \mid t_1 < a_j < t_2 < b_j, e \in p_j\}.$$

Let δ_e denote the difference between the number of arrivals and the number of departures in the time interval $[t_1, t_2)$ among the requests that were routed along e. Clearly $\delta_e \le |\mathit{Alive}_{\in e}(t_1, t_2)|$.

Lemma 4 implies that

$$x_e^{(t_2)} = x_e^{(t_1)} \cdot \left(1 + \frac{1}{4c_e}\right)^{\delta_e}.$$

The assumption that $x_e^{(t_2)} > 3$ and $x_e^{(t_1)} < 1$ imply

$$\left(1 + \frac{1}{4c_e}\right)^{\delta_e} \ge 3\,.$$

Since $1+x \leq e^x$, it follows that $\delta_e > 4 \cdot c_e$. Hence,

$$|\mathit{Alive}_{\in e}(t_1, t_2)| > 4 \cdot c_e \,. \tag{5}$$

By Equation 1, for each $r_j \in \mathit{Alive}_{\in e}(t_1, t_2)$, we have:

$$\Delta_{a_j} P < 1 - \frac{1}{4c_e}. \tag{6}$$

Hence,

$$\begin{aligned} P_{t_2} &= \frac{1}{4m} \cdot m + \sum_{t=0}^{t_2-1} \Delta_t P \\ &= \frac{1}{4} + \sum_{r_j \in \mathit{Dead}_{t_2}} (\Delta_{a_j} P + \Delta_{b_j} P) + \sum_{r_j \in \mathit{Alive}_{t_2}} \Delta_{a_j} P \\ &\leq \frac{1}{4} + \sum_{r_j \in \mathit{Alive}_{t_2}} \Delta_{a_j} P \\ &< \frac{1}{4} + |\mathit{Alive}_{t_2}| - \frac{|\mathit{Alive}_{\in e}(t_1, t_2)|}{4c_e} \\ &< |\mathit{Alive}_{t_2}| \,. \end{aligned} \tag{7}$$

The justification for these lines is as follows. The first line follows from the initialization of the primal variables. The second line follows since every event in time step $t \in [0, t_2 - 1]$ is either an arrival of a request in $\mathit{Dead}_{t_2} \cup \mathit{Alive}_{t_2}$ or a departure of a request in Dead_{t_2}. The third inequality is due to Lemma 5. The fourth equation is due to Equation 6. The last inequality follows from Equation 5.

By Lemma 2, the primal variables at time t_2 are a feasible solution of P-LP(t_2). The optimal value of D-LP(t_2) equals $|\mathit{Alive}_{t_2}|$. Hence, Equation 7 contradicts weak duality, and the lemma follows. □

4.3 Proof of Theorem 1

We now turn to the proof of the main result. The proof is as follows.

Proof of Theorem 1. We begin by proving the bound on the competitive ratio. Lemma 4 states that

$$\forall t \;\; \forall e \in E : x_e = \frac{1}{4m} \cdot \left(1 + \frac{1}{4c_e}\right)^{\mathit{paths}_t(e)} \,.$$

Hence, by Lemma 1, for each original event σ_t,

$$\forall e \in E : \frac{1}{4m} \cdot \left(1 + \frac{1}{4c_e}\right)^{\mathit{paths}_t(e)} \leq 3 \,.$$

Since $2^x \leq 1 + x$ for all $x \in [0, 1]$, it follows that for each original event σ_t

$$\forall e \in E : \mathit{paths}_t(e) \leq c_e \cdot 4 \log(12m) \,,$$

and the first part of the theorem follows.

We now prove the bound on the number of reroutes. Rerouting an alive request r_j occurs if there exists a path $p \in \Gamma_j$ such that $\sum_{e \in p} \frac{x_e}{c_e} < 1 - z_j$. By Line 2 of the ROUTE algorithm, this condition is equivalent to: $\sum_{e \in p} \frac{x_e}{c_e} < \frac{1}{2} \cdot \sum_{e \in p_j} \frac{x_e^{(a_j)}}{c_e}$. Namely, each time a request is rerouted, the weight of the path is at least halved. Note that the halving is with respect to the weight of the path at the time it was allocated.

Let us consider request r_j. Let $p^* \triangleq \text{argmin}_{p \in \Gamma_j}\{\sum_{e \in p} \frac{1}{c_e}\}$. By the choice of a "lightest" path and by Lemma 1, the weight of path p_j is upper bounded by

$$\sum_{e \in p_j} \frac{x_e}{c_e} \leq \sum_{e \in p^*} \frac{x_e}{c_e} \leq 3 \cdot \sum_{e \in p^*} \frac{1}{c_e}\,.$$

By Lemma 4, $x_e \geq 1/(4m)$, hence the weight of path p_j is lower bounded by

$$\sum_{e \in p} \frac{x_e}{c_e} \geq \frac{1}{4m} \cdot \sum_{e \in p} \frac{1}{c_e} \geq \frac{1}{4m} \cdot \sum_{e \in p^*} \frac{1}{c_e}.$$

It follows that the number of reroutes each request undergoes is bounded by $\log_2(12m)$, and the second part of the theorem follows. □

Remark 1. *Note that the first routing request will not be rerouted at all, the second routing request will be rerouted at most twice, and so on. In general, a routing request that arrives at time t will be rerouted at most $|Alive_t|$ times.*

5 Discussion

We present a primal-dual analysis of an online algorithm in a nonmonotone setting. Specifically, we analyze the online algorithm by Awerbuch et al. [AAPW01] for online routing of virtual circuits with unknown durations. We think that the main advantage of this analysis is that it provides an alternative explanation to the stability condition for rerouting that appears in [AAPW01]. According to the primal-dual analysis, rerouting is used simply to preserve the feasibility of the solution of the covering LP.

Our analysis provides a small improvement compared to [AAPW01] in the following sense. The optimal solution in our analysis is both totally flexible (i.e., may reroute every request in every time step) and splittable (i.e., may serve a request using a convex combination of paths). The optimal solution in the analysis of Awerbuch et al. [AAPW01] is only totally flexible and must allocate a path to each request.

The primal-dual approach of Buchbinder and Naor [BN09] is based on bounding the change in the value of the primal solution by the change in the dual solution (this is often denoted by $\Delta P \leq \Delta D$). The main technical challenge we encountered was that this bound simply does not hold in our case. Instead, we use an averaging argument to prove an analogous result (see Lemma 5).

References

[AAF+97] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3):486–504, 1997.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *FOCS '93: Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*, pages 32–40, Washington, DC, USA, 1993. IEEE Computer Society.

[AAPW01] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. *Journal of Computer and System Sciences*, 62(3):385–397, 2001.

[AKP+97] Yossi Azar, Bala Kalyanasundaram, Serge Plotkin, Kirk R Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22(1):93–110, 1997.

[BFGN11] Niv Buchbinder, Moran Feldman, Arpita Ghosh, and Joseph Seffi Naor. Frequency capping in online advertising. In *Algorithms and Data Structures*, pages 147–158. Springer, 2011.

[BN09] Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):99–263, 2009.